SciSpot: Scientific Computing On Temporally Constrained Cloud Preemptible VMs

JCS Kadupitiya, Vikram Jadhao, Prateek Sharma

Abstract—Scientific computing applications are being increasingly deployed on cloud computing platforms. Transient servers such as EC2 spot instances and Google Preemptible VMs, can be used to lower the costs of running applications on the cloud by up to $10\times$. However, the frequent preemptions and resource heterogeneity of these transient servers introduces many challenges in their effective and efficient use. In this paper, we develop techniques for modeling and mitigating preemptions of transient servers, and present SciSpot, a software framework that enables low-cost scientific computing on the cloud. SciSpot deploys applications on Google Cloud Preemptible Virtual Machines that exhibit temporally constrained preemptions: VMs are always preempted in a 24 hour interval. Our empirical analysis shows that the preemption rate is bathtub shaped, which raises multiple fundamental challenges in performance modeling and policy design. We develop a new reliability model for temporally constrained preemptions, and use statistical mechanics to show why the bathtub shape is a fundamental characteristic.

SciSpot's design is guided by our observation that many emerging scientific computing applications that integrate machine learning with simulations, can be deployed as "bags" of jobs, which represent multiple instantiations of the same computation with different physical model parameters. For a bag of jobs, SciSpot finds the optimal transient server on-the-fly, by taking into account the price, performance, and preemption rates of different servers. SciSpot reduces costs by $5\times$ compared to conventional cloud deployments, and reduces makespans by up to $10\times$ compared to conventional high performance computing clusters.

1 Introduction

Increasingly, cloud computing platforms have begun to supplement and complement conventional high performance computing (HPC) infrastructure to meet the large computing and storage requirements of scientific computing applications. Public cloud platforms such as Amazon's EC2, Google Cloud Platform, and Microsoft Azure, offer multiple benefits such as *on-demand* resource allocation, convenient pay-as-you-go pricing models, ease of provisioning and deployment, and near-instantaneous elastic scaling. However, the flexibility offered by cloud platforms comes at a literal cost: the price of deploying scientific computing applications can be significant, and is a major hurdle towards adoption.

Conventionally, cloud VMs have been offered with "on-demand" availability, such that the lifetime of the VM is solely determined by the owner of the VM (i.e., the cloud customer). Increasingly however, cloud providers have begun offering low-cost VMs with *transient*, rather than continuous on-demand availability. Transient VMs can be unilaterally revoked and preempted by the cloud provider, and applications running inside them face fail-stop failures. Due to their volatile nature, transient VMs are offered at steeply discounted rates. Amazon EC2 spot instances, Google Cloud Preemptible VMs, and Azure Batch VMs, are all examples of transient VMs, and are offered at discounts ranging from 50 to 90%.

Deploying applications on such transient VMs requires new mechanisms and policies for mitigating preemptions, which also minimize the total running time and cost. In this paper, we consider a *new* kind of transient availability, which we call *temporally constrained preemptions*. In this availability model, a transient VM has a *fixed* upper bound on its lifetime. Google's Preemptible VMs obey such a model: they have a maximum lifetime of 24 hours, and can be preempted at any point within the $[0, 24]$ hour interval. While these Preemptible VMs are 80% cheaper than their regular non-preemptible counterparts, their effective use requires overcoming several theoretical and practical hurdles, which we seek to achieve.

Temporally Constrained Preemption Challenges. The *first* major challenge lies in understanding and developing a preemption model: when and how frequently do preemptions occur, and what does their distribution look like? Developing such a model is a vital precursor to time and cost minmizing policies, which have been developed for other preemption models (such as for Amazon EC2 spot instances). To this end, we conduct a first-of-its kind empirical study of observing actual VM preemptions, and analyze the resulting data from more than 850 preemption events. Our data-set is open-source and available at [1]. Our findings indicate that due to the temporal constraint, the distribution of preemptions is unlike any other classical failure distribution (such as exponential or Weibull). Instead, the failure distribution is "bathtub" shaped, and has multiple distinct phases: preemptions are more likely to occur at the start and end of the 24 hour interval. This has serious implications for prior transient-computing software, since they all assume exponentially distributed failures, and are unable to efficiently deal with the time-varying failure rates.

The *second* major challenge is developing an analytical model for temporally constrained preemptions. Such a model is essential for characterizing failures and obtaining key reliability metrics such as Mean-Time-To-Failure (MTTF), hazard and survival rate, etc. We therefore develop a simple, generalizable, differentiable analytical model for constrained

This is an expanded and revised version of a preliminary paper that appeared at HPDC 2020 [32].

preemptions, which provides new insights into preemption/-failure dynamics.

Given the prevalence of bathtub-shaped preemptions, the *third* major challenge is to try to understand *why* they occur, and what the fundamental causes could be. For this, we find a surprising connection with *Statistical Mechanics*! We show that preemptions can be "mapped" to the Tonks Gas Model [52], [37], and the bathtub shape arises naturally if we assume the constraint of non-overlapping preemptions.

The *fourth* challenge is reducing the cost and running time of applications deployed on Preemptible VMs. We leverage our analytical preemption model and our analysis, and develop new optimized policies for scheduling and cost-optimization, that mitigate preemptions using insights gleaned from our models.

Finally, we address the practical challenges of deploying scientific batch applications on Preemptible VMs. We identify a new simple abstraction, which we call "Bags of jobs", which is common in many scientific simulation workloads. This abstraction, combined with our optimized policies, allows us to significantly reduce the cost of cloud computing resources, by up to $5\times$. Our resultant system, SciSpot, implements all our policies and abstractions, and provides the first seamless and frictionless service for using Preemptible VMs.

Novelty and Relevance. *Spot markets* (used by Amazon EC2's spot instances and others) are a popular transient computing model, where preemptions are governed by dynamic prices (which are in turn set using a continuous second-price auction [15]). However, the temporally constrained preemption model is distinct from spot markets, and presents fundamental challenges in preemption modeling and effective use of transient VMs. Transiency-mitigation techniques such as VM migration [44], checkpointing [41], [38], diversification [43], *all* use price-signals to model the availability and preemption rates of spot instances. With flat pricing, these approaches are not applicable. Furthermore, no other information about preemption characteristics is made publicly available by the cloud operator, not even coarse-grained metrics, which necessitates our empirical approach.

To expand the usability and appeal of transient VMs, many systems and techniques have been proposed that seek to ameliorate the effects of preemptions and reduce the computing costs of applications. Fault-tolerance mechanisms [44], [38]; resource management policies [43], [55]; and cost optimization techniques [21], [45] have been proposed for a wide range of applications—ranging from interactive web services, distributed data processing, parallel computing, etc. However, these prior works all assume classical exponential failures, which we show to not be ideal for bathtub preemptions.

We make significant analytical, theoretical, and practical enhancements to our preliminary work on understanding constrained preemptions [32]. We provide new empirical insights about preemption dynamics by analyzing the data along a larger number of dimensions. We enhance our analytical model to compute closed-form expressions for MTTF and hazard rate, which are the key reliability metrics used in our policies. We introduce a new "tipping points" based VM scheduling policy which has a key practical benefit of not requiring exact job running times. Finally, we conduct extensive empirical evaluation and show SciSpot's performance and cost, and compare it to the state of the art transient-computing systems.

Contributions. Towards our goal of harnessing temporally constrained transient VMs for modern scientific computing workloads, we make the following contributions:

1) Using a large-scale empirical study of Google's Preemptible VMs, we show a statistical analysis of preemptions based on the VM type, temporal effects, geographical regions, etc. Our analysis indicates that the 24-hour constraint is a defining characteristic, and that the preemption rates are *not* uniform, but have distinct phases.
2) We develop a probability model of constrained preemptions based on empirical and statistical insights that point to distinct failure processes underpinning the preemption rates. Our model captures the key effects resulting from the 24 hour lifetime constraint associated with these VMs.
3) We analyze our probability model through the lens of reliability theory, and demonstrate the use of the principles of statistical mechanics to examine the fundamental behavior of constrained preemptions.
4) Based on our preemption model, we develop optimized policies for job scheduling which minimize the total time and cost of running applications. These policies reduce job running times by up to $2\times$ compared to existing preemption models used for transient VMs.
5) We implement and evaluate our policies as part of a batch computing service for Google Preemptible VMs. SciSpot introduces the bag of jobs abstraction for scientific simulation applications, and can reduce computing costs by $5\times$ compared to conventional cloud deployments, and reduce the performance overhead of preemptible VMs to less than 3%.

2 Background and Related Work

We now give an overview of transient cloud computing, and preemption models.

2.1 Transient Cloud Computing

Infrastructure as a service (IaaS) clouds such as Amazon EC2, Google Public Cloud, Microsoft Azure, etc., typically provide computational resources in the form of virtual machines (VMs), on which users can deploy their applications. Conventionally, these VMs are leased on an "on-demand" basis: cloud customers can start up a VM when needed, and the cloud platform provisions and runs these VMs until they are shut-down by the customer. Cloud workloads, and hence the utilization of cloud platforms, shows large temporal variation. To satisfy user demand, cloud capacity is typically provisioned for the *peak* load, and thus the average utilization tends to be low, of the order of 25% [53], [18].

To increase their overall utilization, large cloud operators have begun to offer their surplus resources as low-cost servers[1]with *transient* availability. These servers can be preempted by the cloud operator at any time (after a small advance warning). These preemptible servers, such as Amazon Spot instances [3], Google Preemptible VMs [6], and Azure batch VMs [13], have become popular in recent years due to their discounted prices, which can be 7-10$\times$ lower than the conventional non-preemptible servers. Due to their popularity among users, smaller cloud providers such as

Packet [7] and Alibaba [2] have also started offering transient cloud servers.

However, effective use of transient servers is challenging for applications because of their uncertain availability [48], [55], [17]. Preemptions are akin to fail-stop failures, and result in loss of the application memory and disk state, leading to downtimes for interactive applications such as web services, and poor throughput for batch-computing applications. Consequently, researchers have explored fault-tolerance techniques such as checkpointing [41], [38], [50] and resource management techniques [43] to ameliorate the effects of preemptions. The effect of preemptions depends on the application's delay insensitivity and fault model, and mitigating preemptions for different applications remains an active research area [31].

2.2 Modeling Preemptions of Transient VMs

The notion of using a probabilistic or even a deterministic model of preemptions underlies *all* techniques and systems in transient computing. Such a preemption model is then used to quantify and analyze the impact of preemptions on application performance and availability; and to design model-informed policies to minimize the effect of preemptions. For example, the preemption rate or MTTF (Mean Time To Failure) of transient servers has found extensive use in selecting the appropriate type of transient server for applications [43], [50], determining the optimal checkpointing frequency [41], [38], [26], [23], etc.

Preemptions of spot market based VMs (such as EC2 spot instances) are based on their *price*, which is dynamically adjusted based on the supply and demand of cloud resources. Spot prices are based on a continuous second-price auction, and if the spot price increases above a pre-specified maximum-price, then the server may be preempted [15]. Thus, the time-series of these spot prices can be used for understanding preemption characteristics such as the frequency of preemptions and the "Mean Time To Failure" (MTTF) of the spot instances. Publicly available [29] historical spot prices have been used to characterize and model spot instance preemptions [44], [60], [46], [56], [57], [54], [51], [24], [27]. For example, past work has analyzed spot prices and shown that the MTTFs of spot instances of different hardware configurations and geographical zones range from a few hours to a few days [58], [39], [56], [14], [57]. Spot instance preemptions can be modeled using *memoryless* exponential distributions [60], [42], [41], [23], [59], which permits optimized periodic checkpointing policies such as Young-Daly [19].

However, using pricing information for preemption modeling is *not* a generalizable approach, and is not applicable to models of transient availability used by other transient VMs like Google Preemptible VMs and Azure Low-priority batch VMs. These VMs have *flat* pricing, and thus pricing cannot be used to infer preemptions. Moreover, these cloud providers (Google and Azure) do not expose *any* public information about their preemption characteristics, even metrics like MTTF that can be useful in mitigating preemptions [59], [47]. In this paper, we propose an empirical approach for modeling preemptions of temporally constrained VMs such as Google Preemptible VMs. Our empirical data and the resulting preemption model allows the development of preemption mitigation policies. Google Preemptible VMs have a maximum lifetime of 24 hours, and this *constrained* preemption is not memoryless, and requires new modeling approaches.

1. We use servers and VMs interchangeably throughout the paper.

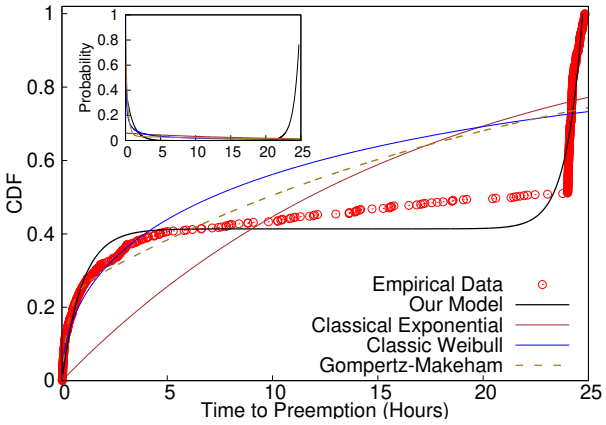


Fig. 1: CDF of lifetimes of Google Preemptible VMs. Our proposed distribution for modeling the constrained preemption dynamics provides a better fit to the empirical data compared to other failure distributions. Inset shows the probability density functions.

3 Preemption Analysis and Modeling of Google Preemptible VMs

In our quest to understand temporally constrained preemptions, we conduct an empirical study of preemptions of Google Preemptible VMs. Based on our observations and insights from the study, we develop a probability model for temporally constrained preemptions, which we then interpret using the framework of statistical mechanics.

3.1 Empirical Study Of Preemptions

Methodology. We launched 870 Google Preemptible VMs of different types over a two month period (Feb–April 2019), and measured their time to preemption (i.e., their useful lifetime). VMs of different resource capacities were launched in four geographical regions; during days and nights and all days of the week; and running different workloads. We launched VMs in their default resource configurations (CPU and memory), and do not use custom VM sizes. To ensure the generality of our empirical observations, VMs were not launched during well-known peak utilization days (such as Black Friday). The preemption data collection was bootstrapped: a small amount of data points were used to estimate and model the preemption CDF, which we then used to run SciSpot (described and evaluated in Sections 5 and 6), which generated the rest of the preemption data. Due to the relatively high preemption rates compared to EC2 spot instances, we were able to collect these data points for less than $5,000.

A sample of over 100 such preemption events are shown in Figure 1, which shows cumulative distribution function (CDF) of the lifetime of the `n1-highcpu-16` VMs in the

us-east1-b zone. The overall rate of preemptions is "bathtub" shaped as shown by the solid black line in the inset of Figure 1. Empirically, our main observations are:

Observation 1: *The lifetimes of VMs are not uniformly distributed, but have three distinct phases.*

In the first (initial) phase, characterized by VM lifetime $t \in [0, 3]$ hours, we observe that many VMs are quickly preempted after they are launched, and thus have a steep rate of failure. The rate of failure (preemption rate) is the derivative of the CDF. The early high rate of failure reflects that the cloud service provider takes into account VM liftetime in prioritizing preempting "younger" VMs, in other words, the number of simultaneous VMs launched does have an effect on their failure rate. In the second phase, VMs that survive past 3 hours enjoy a relatively low preemption rate over a relatively broad range of lifetime (characterized by the slowly rising CDF in Figure 1). The third and final phase exhibits a steep increase in the number of preemptions as the preemption deadline of 24 hours approaches. The overall rate of preemptions is "bathtub" shaped as shown by the solid black line in the inset of Figure 1 (discussed in detail below).

Observation 2: *The preemption behavior, imposed by the constraint of the 24 hour lifetime, is substantially different from conventional failure characteristics of hardware components and EC2 spot instances.*

In "classical" reliability analysis, the rate of failure usually follows an exponential distribution $f(t) = \lambda e^{-\lambda t}$, where $\lambda = 1/\text{MTTF}$. Figure 1 shows the CDF ($= 1 - e^{-\lambda t}$) of the exponential distribution when fitted to the observed preemption data, by finding the distribution parameter λ that minimizes the least squares error. The classic exponential distribution is unable to model the observed preemption behavior because it assumes that the rate of preemptions is independent of the lifetime of the VMs, i.e., the preemptions are *memoryless*. This assumption breaks down when there is a fixed upper bound on the lifetime.

Observation 3: *The three preemption phases and associated bathtub shaped preemption probability, can be seen as general characteristics of Preemptible VMs.*

Our empirical study looked at preemptions of VMs of different sizes (Figure 2a), at different times of the day (Figure 2b), in different geographical zones (Figure 2c), and running different workloads. We also analyzed VMs launched at different days (Figure 3a) and also analyzed the effect of concurrent VM launches (Figure 3b). In all cases, we find that there are three distinct phases associated with the preemption dynamics giving rise to the bathtub-shaped preemption probability.

Observation 4: *Larger VMs have a higher rate of preemptions.*

Figure 2a shows the preemption data from five different types of VMs in the Google Cloud n1-highcpu-{2,4,8,16,32}, where the number indicates the number of CPUs. All VMs are running in the us-central1-c zone. We see that the larger VMs (16 and 32 CPUs) have a higher probability of preemptions compared to the smaller VMs. While this could be simply due to higher demand for larger VMs, it can also be explained from a cluster management perspective. Larger VMs require more computational resources (such as CPU and memory), and when the supply of resources is low, the cloud operator can quickly reclaim a large amount of resources by preempting larger VMs. This observed behavior aligns with the guidelines for using preemptible VMs that suggests the use of smaller VMs when possible [6].

Observation 5: *Preemptions exhibit diurnal variations, and are also affected by the workload inside the VM.*

From Figure 2b, we can see that VMs have a slightly longer lifetime during the night (8 PM to 8 AM) than during the day[2]. This is expected because fundamentally, the preemption rates are higher during periods of higher demand. We also notice that completely idle VMs have longer lifetimes than VMs running some workload, presumably because idle VMs are easier for resource overcommittment [40], [22], and thus have a lower preemption probability.

Significance of bathtub preemptions. The bathtub shaped preemption distribution is not a coincidence. It is a result of fundamental characteristics of constrained preemptions that benefit applications. For applications that do not incorporate explicit fault-tolerance (such as checkpointing), early preemptions result in less wasted work than if the preemptions were uniformly distributed over the 24 hour interval. Furthermore, the low rate of preemptions in the middle periods allows jobs that are smaller than 24 hours to finish execution with only a low probability of failure, once they survive the initial preemption phase. We compare application performance with bathtub preemptions and uniformly distributed preemptions later in Section 6, and find that bathtub preemptions can reduce the performance overheads of preemptions by up to $10\times$. However, effective policies for constrained preemptions require a probability model of preemptions, which must incorporate the temporal constraint and the steep bathtub behavior. Existing preemption models are not applicable, and we present our new model next.

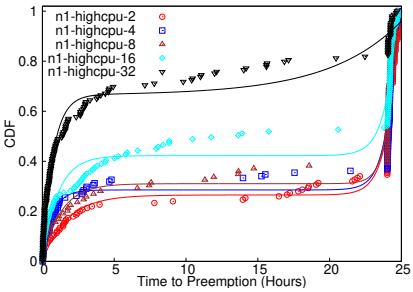
3.2 Failure Probability Model

We now develop an analytical probability model for finding a preemption at a given time that is faithful to the empirically observed data and provides a basis for developing running-time and cost-minimizing optimizations. Modeling the dynamics of preemptions constrained by a finite deadline raises many challenges for existing preemption models that have been used for other transient servers such as EC2 spot instances. We first discuss why existing approaches to preemption modeling are not adequate, and then present our probability model and associated reliability theory connections.

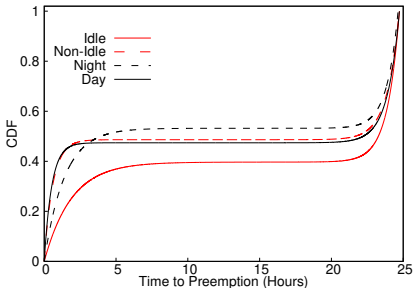
3.2.1 Our model

Our failure probability model seeks to address the drawbacks of existing reliability theory models for modeling constrained preemptions. The presence of three distinct phases exhibiting non-differentiable transition points (sudden changes in CDF near the deadline, for example) suggests that for accurate results, models that treat the probability as a step function (CDF as a piecewise-continuous function) could be employed. However, this limits the range of model applicability and general interpretability of the underlying preemption behavior. Our goal is to provide a broadly applicable, continuously differentiable, and informative model built on reasonable assumptions.

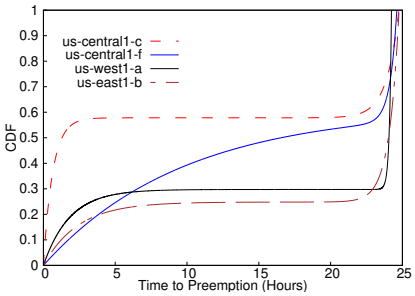
2. Time-zone local to the VM's location.



(a) Preemption characteristics of different VM types. Larger VMs are more likely to be preempted.

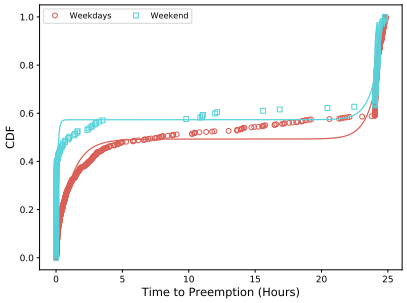


(b) Variations due to time of day and workload.

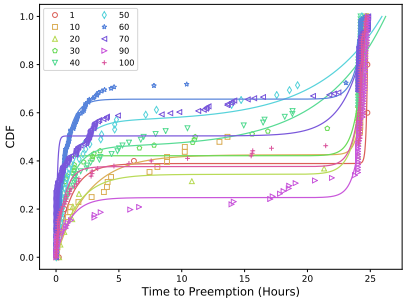


(c) `n1-highcpu-16` in different regions.

Fig. 2: Analysis of preemption characteristics by VM-type, region, time-of-day, and workload type.



(a) By day of week.



(b) Parallel VM launches.

Fig. 3: Preemption CDF analysis.

We begin by making a key assumption: the preemption behavior arises from the presence of *two* distinct failure processes. The first process dominates over the initial temporal phase and yields the classic exponential distribution that captures the high rate of early preemptions. The second process dominates over the final phase near the 24 hour maximum VM lifetime and is assumed to be characterized by an exponential term that captures the sharp rise in preemptions that results from this constrained lifetime.

Based on these observations, we propose the following general form for the CDF:

$$\boxed{\mathscr{F}(t) = A\left(1 - e^{-\frac{t}{\tau_1}} + e^{\frac{t-b}{\tau_2}}\right)} \tag{1}$$

where t is the time to preemption, $1/\tau_1$ is the rate of preemptions in the initial phase, $1/\tau_2$ is the rate of preemptions in the final phase, b denotes the time that characterizes "activation" of the final phase where preemptions occur at a very high rate, and A is a scaling constant. The model is fit to data for $0 < t < L$, where $L \approx 24$ hours represents the temporal interval (deadline). Combination of the 4 fit parameters (τ_1, τ_2, b, and A) are chosen to ensure that boundary condition $\mathscr{F}(0) \approx 0$ is satisfied. In practice, typical fit values yield $b \approx 24$ hours, $\tau_1 \in [0.5, 1, 5]$, $\tau_2 \approx 0.8$, and $A \in [0.4, 0.5]$.

For most of its life, a VM sees failures according to the classic exponential distribution with failure-rate equal to $1/\tau_1$ – this behavior is captured by the $1-e^{-t/\tau_1}$ term in Equation 1. As VMs get closer to their maximum lifetime imposed by the cloud operator, they are reclaimed (i.e., preempted) at a high rate $1/\tau_2$, which is captured by the second exponential term, $e^{(t-b)/\tau_2}$ of Equation 1. Shifting the argument (t) of this term by b ensures that the exponential reclamation is only applicable near the end of the VM's maximum lifetime and does not dominate over the entire temporal range.

The analytical model and the associated distribution function $\mathscr{F}$ introduced above provides a much better fit to the empirical data (Figure 1) compared to other models, and captures the different phases of the preemption dynamics through parameters τ_1, τ_2, b, and A. These parameters can be obtained for a given empirical CDF using least squares function fitting methods (we use scipy's `optimize.curve_fit` with the dogbox technique [8]). The failure or preemption rate can be derived from the CDF in Equation 1 as:

$$f(t) = \frac{d\mathscr{F}(t)}{dt} = A\left(\frac{1}{\tau_1}e^{-t/\tau_1} + \frac{1}{\tau_2}e^{\frac{t-b}{\tau_2}}\right). \tag{2}$$

$f(t)$ vs. t yields a bathtub type failure rate function for the associated fit parameters (inset of Figure 1).

3.2.2 *Reliability Analysis*

We now analyze and place our model in a reliability theory framework.

Expected Lifetime. Our analytical model helps crystallize the differences in VM preemption dynamics, by allowing us to easily calculate their expected lifetime. More formally, we define the expected lifetime of a VM ($\mathscr{L}$) as:

$$E[\mathscr{L}] = \int_0^L t f(t)\, dt = -A(t+\tau_1)e^{-t/\tau_1} + A(t-\tau_2)e^{\frac{t-b}{\tau_2}}\Big|_0^L \tag{3}$$

where $f(t)$ is the rate of preemptions of the VM (Equation 2). This expected lifetime can be used in lieu of MTTF, for policies and applications that require a "coarse-grained"

comparison of the preemption rates of servers of different types, which has been used for cost-minimizing server selection [41].

Hazard Rate. The hazard rate $\lambda(t)$ governs the dynamics of the failure (or survival) processes. It is generally defined as $\lambda(t) = \frac{f(t)}{S(t)}$ and often expressed via the following differential equation (rate law):

$$\frac{dS(t)}{dt} = -\lambda(t)S(t), \tag{4}$$

where $S(t) = 1-F(t)$ is the survival function associated with a CDF $F(t)$, and $f(t) = dF(t)/dt$ is the failure probability function (rate) at time t. The survival function indicates the amount of VMs that have survived at time t. The hazard rate can also be directly expressed in terms of the CDF as follows: $1 - F(t) = \exp \int_0^t -\lambda(x)\, dx$. The exponential distribution has a constant hazard rate λ. The Gompertz-Makeham distribution has an increasing failure rate to account for the increase in mortality, and its hazard rate is accordingly time-dependent and given by $\lambda(t) = \lambda + \alpha e^{\beta t}$.

Since we model multiple failure rates and deadline-induced preemptions, our hazard rate is expected to increase with time. Defining the survival function for our model: $S = 1 - \mathscr{F}$, and using Eq. 4 yields the hazard rate associated with our model:

$$\lambda = \frac{r_1 e^{-r_1 t} + r_2 e^{r_2(t-b)}}{1/A - 1 + e^{-r_1 t} - e^{r_2(t-b)}} \tag{5}$$

where we have introduced $r_1 = 1/\tau_1$, $r_2 = 1/\tau_2$ to denote the rates of preemptions associated with initial and final phases respectively.

Recall that the sharp increase in preemption rate only happens close to the deadline, which means that $b \lesssim L$. Thus, when $0 < t \ll b$, we get $\lambda(t) \approx r_1$, mimicking the hazard rate for the classic exponential distribution. As t approaches and exceeds b (i.e., $b \lesssim t < L$), the increase in the hazard rate due to the second failure process kicks in, accounting for the deadline-induced rise in preemptions. Note that our hazard rate satisfies $\lambda(t) \geq 0$ for $0 < t < L$.

3.3 Statistical mechanics of constrained preemptions

For constrained preemptions, one might expect to see uniformly distributed preemptions with a probability $1/L$ over $[0, L]$. However, as our empirical analysis shows, the preemption distribution is bathtub-shaped. Interestingly, we can show using exact analytical arguments that non-uniform, bathtub distributions are in fact an *emergent* characteristic of systems with constrained preemptions, modulo some assumptions.

Lemma 1. *Consider N randomly distributed preemptions over an interval $[0, L]$. Assume that each preemption takes $w > 0$ time-units to perform, and preemptions cannot overlap, i.e, they occur in a mutually exclusive manner. Then, there exists $\epsilon > 0$ such that $P(L - \epsilon) > \frac{1}{L}$, where $P(t)$ is the probability of finding a preemption at time t.*

Proof. We first make some preliminary remarks and introduce concepts necessary to complete the proof.

Firstly, mutual exclusion of preemptions implies that there is a finite non-zero waiting time $w > 0$ between preemptions. For N preemptions to occur within L interval, evidently, we must have $Nw < L$. Also, while $w > 0$, the time to perform the preemption is generally expected to be much smaller than the total time interval L (i.e., $w \ll L$). N preemptions occupy a "temporal volume" of Nw (volume here represents the one-dimensional volume). We assume that while a preemption may start at $t = 0$, the last preemption must finish by $t = L$. Thus, the amount of free or excluded "temporal volume" available within the constrained system is $L_e = L - w - (N-1)w = L - Nw$. We note that the concept of excluded volume is routinely employed in the analysis of physical systems such as liquids and polymers, where the excluded volume of particles acts as a constraint that gives rise to steric forces and structural changes in material behavior [37], [30], [49].

Secondly, we observe that the system of N preemptions within a constrained deadline of interval L maps *exactly* to a well known and analytically solvable system in classical statistical mechanics, the one-dimensional Tonks gas model [52]. The Tonks gas model describes a system of N non-overlapping particles of finite size w that are constrained to move within a line segment of length L. The structural quantities associated with this system, including the probability of finding a particle at position x within the spatial confinement of length L, are computed by evaluating the partition function of the system, which essentially measures the number of valid system configurations [37]. Employing this mapping, we consider a system of N non-overlapping preemptions constrained within a "time confinement" of size L. Each preemption has access to an excluded volume of L_e within this constrained system. The number of ways N preemptions can occur within the interval L is equivalent to the number of valid configurations for this system, which is given by its partition function: $Z_N = L_e^N = (L - Nw)^N$.

We are interested in calculating the probability that a preemption starts at time $t = L - w$, i.e., $P(L - w)$. Given $w \ll L$, $P(L - w)$ is the probability of finding a preemption near the deadline. The assumption of mutually exclusive preemptions implies that no other preemption can be found for $t > L - w$, that is, $P(t > L - w) = 0$. Hence, the remaining $N - 1$ preemptions must occur such that the last of those finish by $t = L - w$. In other words, the preemption at time $L - w$ essentially sets an effective deadline for the other $N - 1$ preemptions. The number of ways those $N - 1$ preemptions can happen within the time interval of $L - w$ is given by the partition function $Z_{N-1} = L_e^{N-1} = (L - 2w - (N-2)w)^{N-1} = (L - Nw)^{N-1}$, where $L_e = L - Nw$ is the corresponding excluded temporal volume available to each of the $N - 1$ preemptions. It is interesting to note that this excluded volume is the same as that of the original N preemption system: this fortuitous result arises because the decrease in the available volume ("time confinement") to place the preemptions is commensurate with the need to place fewer $(N - 1)$ preemptions.

The probability $P(L - w)$ is obtained as the ratio of the valid configurations given by the two partition functions computed above. That is, $P(L - w) = Z_{N-1}/Z_N = \frac{1}{L - Nw}$. Because $N \geq 1$ and $w > 0$, we find $P(L - w) > \frac{1}{L}$. Choosing $\epsilon = w$ completes the proof. □

By symmetry arguments, the above lemma is in fact

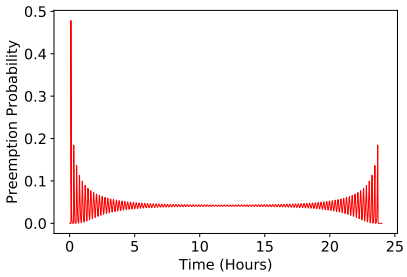


Fig. 4: Preemption probability computed using the partition function defined for a simple constrained system is also bathtub shaped.

valid for both the end points of the interval, i.e., $P(\epsilon) > \frac{1}{L}$. In other words, the probability of preemption is higher near the end points (deadline) than the average preemption probability of $1/L$, and we get a bathtub shaped distribution. For the above proof, we assumed that each preemption event occurs over a timespan of w, which is determined by the preemption warning that the cloud platform provides (which is 30 seconds for Google Preemptible VMs and 120 seconds for Amazon EC2 spot instances). Preempting a VM and reclaiming its resources involves manipulating the cluster-management state, and mutually exclusive preemptions may be convenient for cluster management, since serializing VM preemptions makes accounting and other cluster operations easier. From an application standpoint, non-overlapping preemptions are also beneficial, since handling multiple concurrent preemptions is significantly more challenging [43].

Thus, statistical mechanics indicates that the bathtub distribution follows from the constrained and non-overlapping nature of preemptions, if we assume no other external factors or cloud policies influencing the preemptions. Figure 4 shows the preemption probability computed using the partition function. We find that this probability *also* follows the bathtub shape that is found in the empirical data.

3.4 Model Validity and Generalizability

In the absence of any prior work on constrained preemption dynamics, our aim is to provide an interpretable model with a minimal number of parameters, that provides a sufficiently accurate characterization of features present in the preemption data. Extension of this model to include more failure processes would introduce more parameters and reduce the generalization power. Further, this model as well as the optimization policies derived from it offer pathways to quantify additional effects arising due to the inclusion of more failure processes.

In Section 6.1, we show that constant failure rate assumptions can be severely detrimental to the application failures and performance. We show that our bathtub model, *even with poorly fitted parameters*, can provide significant improvement in job running times, compared to existing exponential models.

4 Policies For Ameliorating Constrained Preemptions

Having analyzed the statistical behavior of constrained preemptions and presented our probability model, we now examine how the bathtub shape of the failure rate impacts applications. Based on insights drawn from our statistical analysis and the model, we develop various policies for ameliorating the effects of preemptions. Prior work in transient computing has established the benefits of such policies for a broad range of applications. However, the constrained nature of preemptions introduces new challenges that do not arise in other transient computing environments such as Amazon EC2 spot instances, and thus new approaches are required. In this section, we first analyze the impact of constrained preemptions on job running time, and then develop new constrained-preemption aware policies for job scheduling and cost optimization. We will focus on long-running batch jobs that arise in many scientific computing applications. Extensions of our models and policies to distributed applications with different failure semantics is part of our future work.

4.1 Job Running Time Modeling

SciSpot uses our bathtub probability model to predict the total running time (i.e., the makespan) of the job. When a preemption occurs during the job's execution, it results in wasted work, assuming there is no checkpointing. This increases the job's total expected running time, since it must restart after a preemption. The expected wasted work depends on two factors:

1) The probability of the job being preempted during its execution.
2) *When* the preemption occurs during the execution.

We can analyze the wasted work due to a preemption using the failure probability model. We first compute the expected amount of wasted work *assuming* the job faces a single preemption, which we denote by $E[W_1(T)]$, where T is the original job running time (without preemptions):

$$E[W_1(T)] = \int_0^T t\, P(t|t \leq T)\, dt. \tag{6}$$

Here, $P(t|t \leq T) = P(t)/P(t \leq T)$. $P(t \leq T)$ is the probability that there is a preemption within time T and is given by $P(t \leq T) = F(T)$, where $F(T)$ is the CDF (Equation 1). $P(t)$ is the probability of a preemption at time t, and is given by $P(t) = f(t)$, where $f(t)$ is the preemption rate (Equation 2). We can therefore write the above equation as:

$$E[W_1(T)] = \int_0^T t\, P(t|t \leq T)\, dt = \frac{1}{F(T)} \int_0^T t\, f(t)\, dt. \tag{7}$$

We note that the integral is the same as the "expected lifetime", given by Equation 3. The above expression for the expected waste due to a single preemption can be used by users and application frameworks to estimate the increase in running time due to preemptions. The total running time (also known as makespan) of a job *with* preemptions is given by:

$$E[T] = P(\text{no failure})\; T + P(1 \text{ failure})\; \left(T + E[W_1(T)]\right), \tag{8}$$

where $P(\text{no failure}) = P(t > T) = 1 - F(T)$ and $P(1 \text{ failure}) = P(t \leq T) = F(T)$. Expanding out these terms and using Equation 7, we get

$$E[T] = (1 - F(T))\, T + F(T)\, (T + E[W_1(T)]) \\ = T + \int_0^T t\, f(t)\, dt. \quad (9)$$

This expression for the expected running time assumes that the job will be preempted at most once. An expression which considers multiple (> 1) job failures easily follows from this base case, but presents relatively low practical value.

Consequences for applications. Based on our analysis, both the increase in wasted time ($E[W_1(T)]/T$) and expected running time ($E[T]/T$) depend on the length of the job for non-memoryless constrained preemptions. For memoryless exponential distributions, the expected waste is simply $T/2$, but this assumption is not valid for constrained preemptions, and thus job lengths must be considered when evaluating the suitability of Preemptible VMs.

Users and transient computing systems can use the expected running time analysis for scheduling and monitoring purposes. Since the preemption characteristics are dependent on the type of the VM and temporal effects, this analysis also allows principled *selection* of VM types for jobs of a given length. For instance, VMs having a higher initial rate of preemptions are particularly detrimental for short jobs, because the jobs will see high rate of failure and are not long enough to run during the VM's stable period with low preemption rates. We evaluate the expected wasted time and running time for Google Preemptible VMs later in Section 6.

4.2 Transition-Points based Job Scheduling and VM Reuse Policy

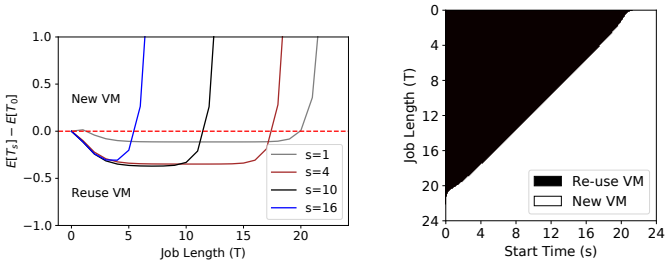
Our bathtub probability model also allows us to develop optimized job-scheduling policies for *reducing* job-failures in the bag of jobs execution model. In the case of deadline-constrained bathtub preemptions, we face a choice: we can either run a new job on an already running VM, or relinquish the VM and run the job on a *new* VM. This choice is important in the case of non-uniform failure rates, since the job's failure probability depends on the "age" of the VM. Because of the bathtub failure distribution, VMs enjoy a long period of low failure rates during the middle of their total lifespan. Thus, it is beneficial to *reuse* VMs for multiple jobs, and relinquishing VMs after every job completion may not be an optimal choice.

However, jobs launched towards the end of VM life face a tradeoff. While they may start during periods of low failure rate, the 24 hour deadline-imposed sharp increase in preemptions poses a high risk of preemptions, especially for longer jobs. The alternative is to discard the VM and run the job on a new VM. However, since newly launched VMs also have high preemption rates (and thus high job failure probability), the choice of running the job on an existing VM vs. a new VM is not obvious.

Our job scheduling policy uses the preemption model to determine the preemption probability of jobs of a given length T. Assume that the running VM's age (time since launch) is s. The intuition is to reuse the VM only if the expected running time is lower, compared to running on a new VM. To compute the expected running time of a job of length T starting at vm-age s, we modify our earlier expression for running time (Equation 9) to:

$$E[T_s] = T + \int_s^{s+T} t\, f(t)\, dt \quad (10)$$

The alternative is to discard the VM and launch a new VM, in which case the expected running time is $E[T_0]$. Our job-scheduling policy is simple: When a job of running time T attempts to start on a VM of age s, if $E[T_s] \leq E[T_0]$, then we run the job on the existing VM. Otherwise, a new VM is launched.



(a) Difference in expected running times vs. job length T for different VM age s. All times are in hours.

(b) Transition boundary depends on job length and when the job starts on the VM.

Fig. 5: Reusing vs. running on a new VM.

Practical Relevance: This policy does not require job running times, only a relative measure if job length is greater or smaller than some threshold or not.

Transition Points Analysis. SciSpot builds on the above insight and determines the job-scheduling and VM-reuse decision. Based on different job lengths within a bag of jobs and the age of the VM, we minimize the total expected running time of a job. For a given job running time (T), and the VM age (s), we can compute $\delta(s,T) = E[T_s] - E[T]$. If $\delta(s,T) \geq 0$, then a new VM is used, as illustrated in Figure 5a. The figure shows that reusing VMs is preferred for shorter job lengths, and the range of jobs for which reusing is good depends on the current age of the VM: younger VMs ($s = 1$ hour) can be used for longer job lengths compared to older VMs. For a given VM type, we precompute this function for different (s,T) values, and find the job length transition points where launching a new VM is suitable. The precomputed $\delta(s,T)$ can be seen in Figure 5b. Using this figure, SciSpot, and users in general, can simply lookup the job and VM attributes, and determine the optimal decision.

4.3 Cost-aware VM Selection

VM-selection is an important optimization in cloud environments, because VMs have different tradeoffs of cost, performance, and preemption characteristics. Application performance is affected by the size of the VM (due to network communication and parallel scaling overheads), and the preemption rates. Our policy selects the "right" type of VMs that minimizes the expected job failure probability and cost by using the analytical preemption models.

We assume that the total resource requirement for a job, $\mathcal{R}$, is provided by the user based on prior speedup data, the

user's cloud budget, and the deadline for job completion. Assume that the cloud provider offers N server types, with the price (per unit time) of a server type equal to c_i. The overall expected cost of running a job is:

$$E[C_{(i,n_i)}] = n_i \times c_i \times E[\mathcal{T}_{(i,n_i)}]. \quad (11)$$

Here, $E[\mathcal{T}_{(i,n_i)}]$ denotes the expected makespan of the job (accounting for preemptions) on n_i servers of type i. This turnaround time depends on whether the job needs to be recomputed because of preemptions, and is given by Equation 9. SciSpot searches over all available and acceptable VM types (modulo any resource constraints on minimum memory/CPUs/GPUs), and picks the VM type which yields the lowest expected cost. The search process is aided by the bag of jobs abstraction: initial jobs are used for determining the lowest-cost VM, on which the remaining jobs are run.

5 SciSpot Design and Implementation

We have developed SciSpot as a prototype batch computing service that implements various policies for constrained preemptions. We use it to examine the effectiveness and practical utility of our model and policies in real-world settings. SciSpot is implemented as a light-weight, extensible framework that makes it convenient and cheap to run batch jobs in the cloud. We have implemented our prototype in Python in about 2,000 lines of code, and currently support running VMs on the Google Cloud Platform [5].

We use a centralized controller (Figure 6), which implements the VM selection and job scheduling policies described in Section 4. The controller can run on any machine (including the user's local machine, or inside a cloud VM), and exposes an HTTP API to end-users. Users submit either a bag or individual jobs to the controller via the HTTP API, which then launches and maintains a cluster of cloud VMs, and maintains the job queue and metadata in a local database.

SciSpot integrates, and interfaces with two primary services. First, it uses the Google cloud API [4] for launching, terminating, and monitoring VMs. Once a cluster is launched, it then configures a cluster manager such as Slurm [9] or Torque [10], to which it submits jobs. SciSpot uses the Slurm cluster manager, with each VM acting as a Slurm "cloud" node, which allows Slurm to gracefully handle VM preemptions. The Slurm master node runs on a small, 2 CPU non-preemptible VM, which is shared by all applications and users. We monitor job completions and failures (due to VM preemptions) through the use of Slurm call-backs, which issue HTTP requests back to the central service controller.

Policy Implementation. SciSpot creates and manages clusters of transient cloud servers, manages all aspects of the VM lifecycle and costs, and implements the model-based policies. It manages a cluster of VMs, and parameterizes the bathtub model based on the VM type, region, time-of-day, and day-of-week. When a new batch job is to be launched, SciSpot finds a "free" VM in the cluster that is idle, and uses the job scheduling policy to determine if the VM is suitable or a new VM must be launched. Due to the bathtub nature of the failure rate, VMs that have survived the initial failures are "stable" and have a very low rate of failure, and thus are "valuable". We keep these stable VMs as "hot spares" instead of terminating them, for a period of one hour.

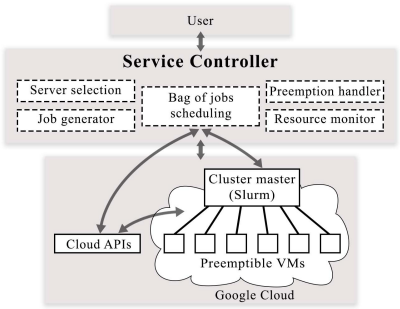


Fig. 6: Architecture and system components of SciSpot.

Bag of Jobs Abstraction For Scientific Simulations. While SciSpot is intended for general batch jobs, we incorporate a special optimization for scientific simulation workloads that improves the ease-of-use of our service, and also helps in our policy implementation. We allow users to submit entire bags of jobs, which permits us to determine the running time of jobs based on previous jobs in the bag. For constrained preemptions, the running time is determined by job lengths, and the job run time estimates are extremely useful. Having a large sequence of jobs is also particularly useful with bathtub-shaped preemption rates, because we can re-use "stable" VMs with low preemption probability for running new jobs from a bag. If jobs were submitted one at a time, a batch computing service may have to terminate the VM after job completion, which would increase the job failure probability resulting from running on new VMs that have a high initial failure rate.

6 SciSpot Experimental Evaluation

In this section, we present analytical and empirical evaluation of SciSpot. We have already presented the statistical analysis of our model in Section 3, and we now focus on answering the following questions:

1) How do constrained preemptions impact the total running time of applications?
2) What is the effect of our model-based policies when compared to existing transient computing approaches?
3) What is the cost and performance of SciSpot for real-world workloads?

Environment and Workloads: All our empirical evaluation is conducted on the Google Cloud Platform using our batch computing service described in Section 5. All the experiments are conducted in the same time period, and have the same preemption characteristics, as described in our data collection methodology in Section 3. We use three scientific computing workloads that are representative of applications in the broad domains of physics and materials science:

Nanoconfinement. The nanoconfinement application launches molecular dynamics (MD) simulations of ions in nanoscale confinement created by material surfaces [30], [34], [12]. The running time is 14 minutes on a 64 CPU core cluster (4 n1-highcpu-16 VMs).

Shapes. The Shapes application runs an MD-based simulated annealing procedure to predict the optimal shape of deformable nanoparticles [28], [16], [33]. The running time is 9 minutes on a 64 CPU core cluster (4 n1-highcpu-16 VMs).

LULESH. Livermore Unstructured Lagrangian Explicit Shock Hydrodynamics (LULESH) is a popular benchmark

for hydrodynamics simulations of continuum models of materials [35], [36]. The running time is 12.5 minutes on 8 n1-highcpu-8 VMs.

We now evaluate the effectiveness of model-driven policies proposed in Section 4. Wherever applicable, we compare against policies designed for EC2 spot instances [25], [50] that have memoryless preemptions. However, we also note that certain resource management challenges such as the preemption-rate aware job scheduling are *inherent* to constrained preemptions, and no existing equivalent policies can be found for memoryless techniques.

6.1 Job Scheduling

In many scenarios, a server may be used for running a long-running sequence of jobs, such as in a batch-computing service. SciSpot's job scheduling policy is model-driven and decides whether to request a new VM for a job or run it on an existing VM. A new VM may be preferable if the job starts running near the VM's 24 hour preemption deadline.

Figure 7 shows the effect our job scheduling policy for a six hour job, for different job starting times (relative to the VM's starting time). We compare against a baseline of memoryless job scheduling that is not informed by constrained preemption dynamics. Such memoryless policies are the default in existing transient computing systems such as SpotOn [50]. In the absence of insights about bathtub-shaped preemption probability, the memoryless policy continues to run jobs on the existing VM. As the figure shows, the empirical job failure probability is bathtub shaped. However, because the job is 6 hours long, it will always fail with the memoryless policy when launched after $24-6=18$ hours. In contrast, SciSpot's model-based policy determines that after 18 hours, we will be better off running the job on a newer VM, and results in a constant lower job failure probability ($=0.4$). The failure probability is constant because the jobs will always be launched on a new VM after 18 hours, resulting in a failure probability at time $=0$. Thus, SciSpot can reduce job failure probability by taking into account the time-varying failure rates of VMs, which is not considered by existing systems that use memoryless scheduling policies.

The job failure probability is determined by the job length and the job starting time. We examine the failure probability for jobs of different lengths (uniformly distributed) in Figure 8, in which we average the failure probability across different start times. We again see that our policy results in significantly lower failure probability compared to memoryless scheduling. For all but the shortest and longest jobs, the failure probability with our policy is *half* of that of existing memoryless policies. This reduction is primarily due to how the two policies perform for jobs launched near the end of the VM preemption deadline, which we examined previously in Figure 7.

Sensitivity to model fitting. The effectiveness of any model-based policy depends on the goodness of fit of the preemption model—i.e., how accurately it captures empirical data. We now evaluate the impact on our scheduling policy if incorrect/suboptimal model parameters with high goodness-of-fit (r^2) error are used. That is, we seek to understand how sensitive our policies are when the underlying preemption behavior does not match the model, which can occur due to changes in supply/demand, minor cloud policy changes, etc. Figure 9 compares the job failure probability with the optimal bathtub model that best fits the empirical data, and a suboptimal bathtub model intentionally chosen to have a bad fit. Specifically, the suboptimal case models the `n1-highcpu-16` VMs for `n1-highcpu-32` VMs, which from Figure 2a we can see are significantly different. However, even with the suboptimal model, the increase in job failure probability is less than 2% compared to the best-fit model. This negligible difference is due to the fact that even a suboptimal model captures the bathtub shape, and this is enough for the policy to make the "right" scheduling decision.

Result: *SciSpot's policies are not particularly sensitive to the exact model parameters, so long as a bathtub distribution is used. Even a suboptimal bathtub model can reduce failure probability by 15% compared to the memoryless policy.*

6.2 Impact of VM Selection

When an application (i.e., bag of jobs) requests a total number of CPUs to run each of its jobs, SciSpot first runs its exploration phase to find the "right" VM for the application. SciSpot searches for the VM that minimizes the total expected cost $E[C_{(i,n_i)}]$ of running the application. Thus, even if the *total* amount of resources (i.e., number of CPUs) per job is held constant, the total running time (i.e., turnaround time) of an application depends on the choice of the VM type (i), and the associated number of VMs (n_i) required to meet the allocation constraint.

Figure 10 shows the running times of the Nanoconfinement, Shapes, and LULESH applications, when they are deployed on different VM sizes. In all cases, the total number of CPUs per job is set to 64, and thus the different VM sizes yield different cluster sizes (e.g., 16 VMs with 4 CPUs or 32 VMs with 2 CPUs). LULESH requires CPUs to be cube of an integer, which limits the valid cluster configurations.

For Nanoconfinement and LULESH, we observe that the base running times (without preemptions) reduce when moving to larger VMs, because this entails lower communication costs. For Nanoconfinement, the running time on the "best" VM (i.e., with 32 CPUs) is nearly 40% lower as compared to the worst case. On the other hand, the Shapes application can scale to a larger number of VMs without any significant communication overheads, and does not see any significant change in its running time.

Figure 10 also shows the expected recomputation time which depends on the expected lifetimes of the VM and the number of VMs. While selecting larger VMs may reduce communication overheads and thus improve performance, it is not an adequate policy in the case of preemptible VMs, since the preemptions can significantly increase the turnaround time. Therefore, even though the base running time of Nanoconfinement is lower on a 64 CPU VM, the recomputation time on the 64 CPU VM is almost $4\times$ higher compared to a 2x32-CPU cluster, due to the much lower expected lifetime of the larger VMs. Thus, on preemptible servers, there is a tradeoff between the base running time which only considers parallelization overheads, and the recomputation time. By considering *both* these factors, SciSpot's server selection policy can select the best VM for an application.

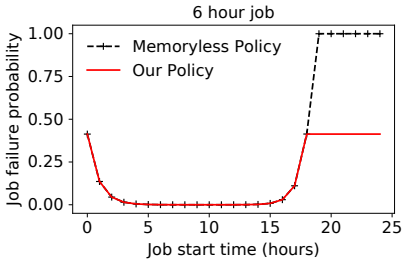


Fig. 7: Effect of job start time on the failure probability.

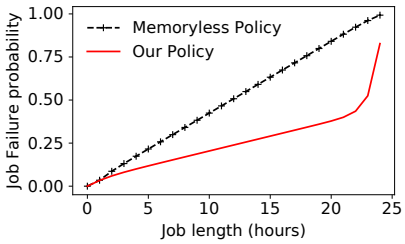


Fig. 8: Job failure probability for jobs of different lengths.

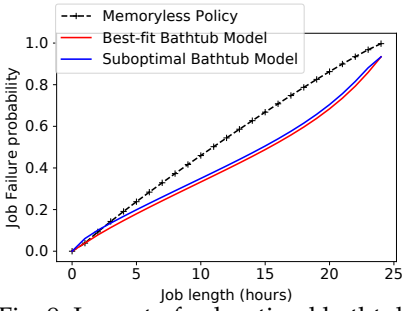


Fig. 9: Impact of suboptimal bathtub model parameters on the scheduling policy is negligible.

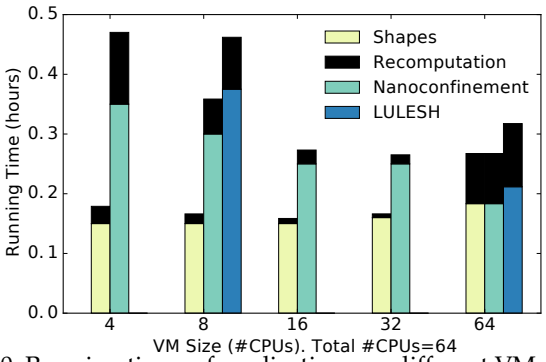


Fig. 10: Running times of applications on different VMs. Total number of CPUs is 64, yielding different number of VMs in each case. We see different tradeoffs in the base running times and recomputation times.

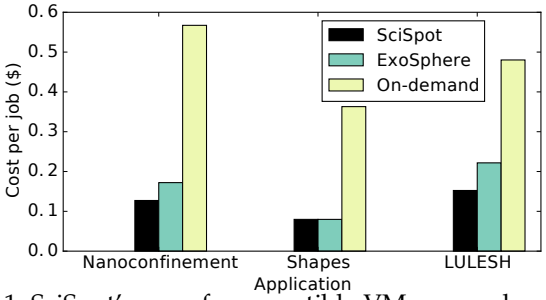


Fig. 11: SciSpot's use of preemptible VMs can reduce costs by up to 5× compared to conventional cloud deployments, and 20% compared to the state of the art EC2 spot instance selection (ExoSphere [43]).

Result: *SciSpot's server selection, by considering both the base running time and recomputation time, can improve performance by up to 40%, and can keep the increase in running time due to preemptions to less than 5%.*

6.3 Cost

The primary motivation for using preemptible VMs is their significantly lower cost compared to conventional "on-demand" cloud VMs that are non-preemptible. Figure 11 compares the cost of running different applications with different cloud VM deployments. SciSpot, which uses both cost-minimizing server selection, and preemptible VMs, results in significantly lower costs across the board, even when accounting for preemptions and recomputations. We also compare against ExoSphere [43], a state of the art system for transient server selection. ExoSphere implements a portfolio-theory approach using EC2 spot prices to balance average cost saving and risk of revocations using diversification and selecting VMs with low price correlation. However, this approach is ineffective for the flat prices of Google Preemptible VMs. Unlike SciSpot, ExoSphere does *not* consider application performance when selecting servers, and thus is unable to select the best server for parallel applications. Since the Google `highcpu` VMs have the same price per CPU, ExoSphere picks an arbitrary "median" VM to break ties, which may not necessarily yield the lowest running times. This results in 20% cost increase over SciSpot.
Result: *SciSpot reduces computing costs by up to 5× compared to conventional on-demand cloud deployments.*

6.4 Comparison with HPC Overhead

Scientific computing applications are typically run on large-scale HPC clusters, where different performance and cost dynamics apply. While there are hardware differences between cloud VMs and HPC clusters that can contribute to performance differences, we are interested in the performance "overheads". In the case of SciSpot, the job failures and recomputations increase the job turnaround time, and are thus the main source of overhead.

On HPC clusters, jobs enjoy significantly lower recomputation probability, since the hardware on these clusters has MTTFs in the range of years to centuries [20]. However, we emphasize that there exist *other* sources of performance overheads in HPC clusters. In particular, since HPC clusters have high resource utilization, they also have significant *waiting* times. On the other hand, cloud resource utilization is low [53] and there is usually no need to wait for resources, which is why transient servers exist in the first place.

Thus, we compare the performance overhead due to preemptions for SciSpot, and job waiting times in conventional HPC deployments. To obtain the job waiting times in HPC clusters, we use the LANL Mustang traces published as part of the Atlas trace repository [11]. We analyze the waiting time of over two million jobs submitted over a 5 year period, and compute the increase in running time of the job due to the job waiting or queuing time.

Figure 12 compares the overhead (as percentage increase in running time) of SciSpot and HPC clusters for jobs of different lengths. We see that the average performance overhead due to waiting can be significant in the case of HPC clusters, and the job submission latency and queuing time dominate for smaller jobs, increasing their total turnaround

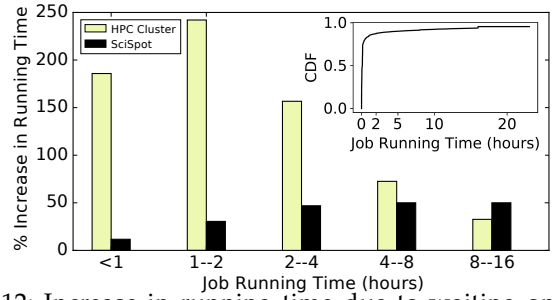


Fig. 12: Increase in running time due to waiting on HPC clusters is significantly higher than the recomputation time for SciSpot, except for very long and rare jobs (see inset).

time by more than $2.5\times$. This waiting is amortized in the case of longer running jobs, and the overhead for longer jobs is around 30%.

On the other hand, SciSpot's performance overhead is significantly smaller for jobs of up to 8 hours in length. For longer jobs, the limited lifetime of Google Preemptible VMs (24 hours) begins to significantly increase the preemption probability and expected recomputation time. We emphasize that these are *individual* job lengths, and not the running time of entire bag of jobs. We note that these large single jobs are rare, accounting for less than 5% of all HPC jobs (see inset in Figure 12). For smaller jobs (within a much larger bag), both the preemption probability and recomputation overhead is much smaller.

Result: *SciSpot's overhead of recomputation due to preemptions is small, and is up to* $10\times$ *lower compared to the overhead of waiting in conventional HPC clusters.*

7 Discussion and Future Directions

Constrained preemptions are a relatively unexplored phenomenon and challenging to model. Our model and the associated data expand transient cloud computing to beyond EC2-spot. However, many questions and avenues of future investigation remain open:

What if preemption characteristics change? Our model allows detecting policy and phase changes by comparing observed data with model-predictions and detect change-points, and a long-running cloud service can continuously update the model based on recent preemption behavior. However, changes are rare: Google's preemption policy has not changed since its inception in 2015. We have also shown that our policies are not particularly sensitive to the model parameters, and even using a "wrong" or outdated model can provide significant benefits compared to existing memoryless models. Our modeling approach works across a wide range of instance types and is able to model CDFs of instances with both very high and very low failure rates, and thus is general. Moreover, because bathtub preemptions are good for the applications, they will continue to remain a good choice for constrained preemptions making our approach generalizable to other system environments beyond the Google cloud computing systems. Finally, the principle adopted to break down the problem into the superposition of processes characterized by different failure rates can also be considered as a general framework to understand and guide policies for mitigating preemption-induced effects in other cloud environments.

For robust long-term relevance of the preemption model, we envision a community-driven approach. An increase in SciSpot use by the research community will provide more preemption data, which can then be used to constantly refine the model when preemption characteristics change due to cloud policies or supply/demand fluctuations.

Generalizability to other Transient VMs. In this paper, we focus on constrained preemptions. Other cloud transient VMs such as spot instances do not have the temporal constraint, and their preemption modeling can be performed with classical distributions (such as exponential distribution for modeling EC2 spot instance preemptions [60], [41], [42]). However, we have shown that an empirical approach of collecting actual preemption data is feasible and effective, and a principled approach can be used even without spot-market price signals.

8 Conclusion

The effective use of transient computing relies on understanding the preemption characteristics. While past work on transient computing has developed techniques and systems for Amazon's EC2 spot instances, ours is the *first* work to understand the behavior of Google's Preemptible VMs, that have a unique characteristic of having a maximum 24 hour lifetime. Our large-scale empirical study shows that the constraint imposes a bathtub failure distribution, and we develop a new preemption probability model for capturing its three distinct temporal phases. Our insights and model-based policies can reduce the preemption overheads by more than $5\times$ compared to existing preemption models, and our batch computing service can reduce computing costs by over $5\times$.

References

[1] https://github.com/kadupitiya/goog-preemption-data/.

[2] Alibaba Cloud Preemptible Instances. https://www.alibabacloud.com/help/doc-detail/52088.htm .

[3] Amazon EC2 Spot Instances, howpublished=https://aws.amazon.com/ec2/spot/ ,.

[4] Google Cloud API Documentation. https://cloud.google.com/apis/docs/overview .

[5] Google Cloud Platform. https://cloud.google.com/ .

[6] Google Cloud Preemptible VM Instances Documentation, howpublished=https://cloud.google.com/compute/docs/instances/preemptible,.

[7] Packet Spot Market. https://support.packet.com/kb/articles/spot-market .

[8] Scipy curve fit documentation. https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.curve$_fit$.

[9] Slurm Workload Manager. https://slurm.schedmd.com/documentation.html .

[10] Torque Resource Manager. http://www.adaptivecomputing.com/products/torqu

[11] G. Amvrosiadis, J. W. Park, G. R. Ganger, G. A. Gibson, E. Baseman, and N. DeBardeleben. On the diversity of cluster workloads and its impact on research results. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 533–546, 2018.

[12] N. Anousheh, F. J. Solis, and V. Jadhao. Ionic structure and decay length in highly concentrated confined electrolytes. *AIP Advances*, 10(12):125312, 2020.

[13] Azure Low-priority Batch VMs. https://docs.microsoft.com/en-us/azure/batch/batch-low-pri-vms .

[14] M. Baughman, C. Haas, R. Wolski, I. Foster, and K. Chard. Predicting amazon spot prices with lstm networks. In *Proceedings of the 9th Workshop on Scientific Cloud Computing*, page 1. ACM, 2018.

[15] O. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing Amazon EC2 Spot Instance Pricing. *ACM TEC*, 1(3), September 2013.

[16] N. E. Brunk and V. Jadhao. Computational studies of shape control of charged deformable nanocontainers. *Journal of Materials Chemistry B*, 7:6370, 2019.

[17] N. Chohan, C. Castillo, M. Spreitzer, M. Steinder, A. Tantawi, and C. Krintz. See Spot Run: Using Spot Instances for MapReduce Workflows. In *HotCloud*, June 2010.

[18] E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini. Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, pages 153–167, New York, NY, USA, 2017. ACM.

[19] J. T. Daly. A Higher Order Estimate of the Optimum Checkpoint Interval for Restart Dumps. *Future Generation Computer Systems*, 22(3), 2006.

[20] J. Dongarra, T. Herault, and Y. Robert. Fault tolerance techniques for high-performance computing. page 66.

[21] D. J. Dubois and G. Casale. Optispot: minimizing application deployment cost using spot cloud resources. *Cluster Computing*, pages 1–17, 2016.

[22] A. Fuerst, A. Ali-Eldin, P. Shenoy, and P. Sharma. Cloud-scale VM-deflation for Running Interactive Applications On Transient Servers. In *HPDC*, 2020.

[23] B. Ghit and D. Epema. Better safe than sorry: Grappling with failures of in-memory data analytics frameworks. In *Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '17, pages 105–116, New York, NY, USA, 2017. ACM.

[24] W. Guo, K. Chen, Y. Wu, and W. Zheng. Bidding for Highly Available Services with Low Price in Spot Instance Market. In *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing - HPDC '15*, pages 191–202, Portland, Oregon, USA, 2015. ACM Press.

[25] A. Harlap, A. Chung, A. Tumanov, G. R. Ganger, and P. B. Gibbons. Tributary: spot-dancing for elastic services with latency slos. In *2018 {USENIX} Annual Technical Conference ({USENIX}{ATC} 18)*, pages 1–14, 2018.

[26] A. Harlap, A. Tumanov, A. Chung, G. R. Ganger, and P. B. Gibbons. Proteus: Agile ml elasticity through tiered reliability in dynamic resource markets. In *Proceedings of the Twelfth European Conference on Computer Systems*, EuroSys '17, pages 589–604, New York, NY, USA, 2017. ACM.

[27] D. Irwin, P. Shenoy, P. Ambati, P. Sharma, S. Shastri, and A. Ali-Eldin. The price is (not) right: Reflections on pricing for transient cloud servers. *2019 28th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, 2019.

[28] V. Jadhao, C. K. Thomas, and M. Olvera de la Cruz. Electrostatics-driven shape transitions in soft shells. *Proceedings of the National Academy of Sciences*, 111(35):12673–12678, 2014.

[29] B. Javadi, R. Thulasiram, and R. Buyya. Statistical Modeling of Spot Instance Prices in Public Cloud Environments. In *UCC*, December 2011.

[30] Y. Jing, V. Jadhao, J. W. Zwanikken, and M. Olvera de la Cruz. Ionic structure in liquids confined by dielectric interfaces. *The Journal of chemical physics*, 143(19):194508, 2015.

[31] P. Joaquim, M. Bravo, L. Rodrigues, and M. Matos. Hourglass: Leveraging transient resources for time-constrained graph processing in the cloud. In *Proceedings of the Fourteenth EuroSys Conference 2019*, EuroSys '19, pages 35:1–35:16, New York, NY, USA, 2019. ACM.

[32] J. Kadupitige, V. Jadhao, and P. Sharma. Modeling the temporally constrained preemptions of transient cloud vms. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, page 4152, New York, NY, USA, 2020. Association for Computing Machinery.

[33] J. Kadupitiya, N. Brunk, and V. Jadhao. Nanoparticle shape lab, January 2020. nanoHUB.

[34] J. Kadupitiya, S. Marru, G. C. Fox, and V. Jadhao. Ions in nanoconfinement, Dec 2017. Online on nanoHUB; source code on GitHub at github.com/softmaterialslab/nanoconfinement-md.

[35] I. Karlin, A. Bhatele, J. Keasler, B. L. Chamberlain, J. Cohen, Z. DeVito, R. Haque, D. Laney, E. Luke, F. Wang, D. Richards, M. Schulz, and C. Still. Exploring traditional and emerging parallel programming models using a proxy application. In *27th IEEE International Parallel & Distributed Processing Symposium (IEEE IPDPS 2013)*, Boston, USA, May 2013.

[36] I. Karlin, J. Keasler, and R. Neely. Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973, August 2013.

[37] W. Krauth. *Statistical mechanics: algorithms and computations*, volume 13. OUP Oxford, 2006.

[38] A. Marathe, R. Harris, D. Lowenthal, B. R. De Supinski, B. Rountree, and M. Schulz. Exploiting redundancy for cost-effective, time-constrained execution of hpc applications on amazon ec2. In *HPDC*. ACM, 2014.

[39] X. Ouyang, D. Irwin, and P. Shenoy. Spotlight: An information service for the cloud. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*, 2016.

[40] P. Sharma, A. Ali-Edlin, and P. Shenoy. Resource Deflation: A New Approach For Transient Resource Reclamation. In *EuroSys*, 2019.

[41] P. Sharma, T. Guo, X. He, D. Irwin, and P. Shenoy. Flint: Batch-Interactive Data-Intensive Processing on Transient Servers. In *EuroSys*, April 2016.

[42] P. Sharma, D. Irwin, and P. Shenoy. How Not to Bid the Cloud. In *Proceedings of the 8th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*. USENIX, June 2016.

[43] P. Sharma, D. Irwin, and P. Shenoy. Portfolio-driven resource management for transient cloud servers. In *Proceedings of ACM Measurement and Analysis of Computer Systems*, volume 1, page 23, June 2017.

[44] P. Sharma, S. Lee, T. Guo, D. Irwin, and P. Shenoy. SpotCheck: Designing a Derivative IaaS Cloud on the Spot Market. In *EuroSys*, April 2015.

[45] S. Shastri and D. Irwin. Hotspot: automated server hopping in cloud spot markets. In *Proceedings of the 2017 Symposium on Cloud Computing*, pages 493–505. ACM, 2017.

[46] S. Shastri, A. Rizk, and D. Irwin. Transient guarantees: Maximizing the value of idle cloud capacity. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '16, pages 85:1–85:11, Piscataway, NJ, USA, 2016. IEEE Press.

[47] R. W. Shijian Li and T. Guo. Characterizing and modeling distributed training with transient cloud gpu servers. *40th IEEE International Conference on Distributed Computing Systems (ICDCS'20)*.

[48] R. Singh, P. Sharma, D. Irwin, P. Shenoy, and K. Ramakrishnan. Here Today, Gone Tomorrow: Exploiting Transient Servers in Data Centers. *IEEE Internet Computing*, 18(4), July/August 2014.

[49] F. J. Solis, V. Jadhao, and M. O. De La Cruz. Generating true minima in constrained variational formulations via modified lagrange multipliers. *Physical Review E*, 88(5):053306, 2013.

[50] S. Subramanya, T. Guo, P. Sharma, D. Irwin, and P. Shenoy. SpotOn: A Batch Computing Service for the Spot Market. In *SOCC*, August 2015.

[51] S. Tang, J. Yuan, and X. Li. Towards Optimal Bidding Strategy for Amazon EC2 Cloud Spot Instance. In *CLOUD*, June 2012.

[52] L. Tonks. The complete equation of state of one, two and three-dimensional gases of hard elastic spheres. *Phys. Rev.*, 50:955–963, Nov 1936.

[53] A. Verma, L. Pedrosa, M. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes. Large-scale cluster management at google with borg. In *EuroSys*. ACM, 2015.

[54] S. Wee. Debunking Real-Time Pricing in Cloud Computing. In *CCGrid*, May 2011.

[55] A. Wieder, P. Bhatotia, A. Post, and R. Rodrigues. Orchestrating the deployment of computations in the cloud with conductor. In *NSDI 12*, 2012.

[56] R. Wolski and J. Brevik. Providing statistical reliability guarantees in the aws spot tier. In *Proceedings of the 24th High Performance Computing Symposium*, page 13. Society for Computer Simulation International, 2016.

[57] R. Wolski, J. Brevik, R. Chard, and K. Chard. Probabilistic guarantees of execution duration for amazon spot instances. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, page 18. ACM, 2017.

[58] R. Wolski, J. Brevik, R. Chard, and K. Chard. Probabilistic guarantees of execution duration for Amazon spot instances. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis on - SC '17*, pages 1–11, Denver, Colorado, 2017. ACM Press.

[59] C. Zhang, V. Gupta, and A. A. Chien. Information models: Creating and preserving value in volatile cloud resources. In *2019 IEEE International Conference on Cloud Engineering (IC2E)*, pages 45–55, June 2019.

[60] L. Zheng, C. Joe-Wong, C. W. Tan, M. Chiang, and X. Wang. How to Bid the Cloud. In *SIGCOMM*, August 2015.

JCS Kadupitiya is a Ph.D. candidate in the Department of Intelligent Systems Engineering at Indiana University Bloomington. He received a Master's in Computer Engineering from Indiana University Bloomington in 2019. His research interests lie primarily in scientific machine learning and distributed computing. He completed his Bachelors degree in Electrical and Information Engineering from the University of Ruhuna, Sri Lanka, in 2015. He also received a Master's in Computer Science and Engineering at the University of Moratuwa, Sri Lanka, in 2017.

Vikram Jadhao is an assistant professor of Intelligent Systems Engineering at Indiana University in Bloomington. Prior to joining Indiana University, he held postdoctoral fellowships in the Department of Physics and Astronomy at Johns Hopkins University as well as the Department of Materials Science and Engineering at Northwestern University. He received his Ph.D. and M.S. in Physics from the University of Illinois at Urbana-Champaign, and his B.S. in Physics from the Indian Institute of Technology at Kharagpur. His research interests are broadly in computational modeling and simulation of soft materials at the nanoscale.

Prateek Sharma is an an assistant professor of Intelligent Systems Engineering at Indiana University in Bloomington. He has a PhD in Computer Science from the University of Massachusetts Amherst. His current research focuses on Cloud Computing, Operating Systems, and Virtualization. Sharma received his masters degree in Computer Science from Indian Institute of Technology, Bombay. Contact him at prateeks@iu.edu.